Symbolic Machine Translation Models

Alan Jaffe

March 22, 2017

1 Baseline

The goal of this project was to implement a symbolic machine translation model for translating from German to English using the IWSLT data. I began by implementing the baseline, a phrase-based translation model based on the noisy channel model. IBM model 1 was used to find alignments, and phrases were extracted from these alignments. A language model for English was generated using an ngram model with linear interpolation between bigrams and unigrams, which was trained using maximum likelihood estimation. OpenFST was used to find the minimum cost translation that combined the translation cost with the language model cost.

2 Synchronization of Aligments

I initially used only one-to-many alignments. However, I noticed that many of the phrases generated were not meaningful phrases, instead including extra unrelated words (e.g. "unserem" was paired with "our collective well-being" when it should just mean "our"). This occurred because many words did not have any alignment and thus the phrase extraction algorithm had too much leniency to construct spurious phrases. Preventing the IBM model from generating NULL alignments in its final output (while still allowing NULL alignments during the EM training) seemed to improve translation accuracy and reduce the number of spurious phrases.

I also attempted to apply the union method for synchronization of alignments. This was successful in drastically reducing the number of spurious phrases generated (by increasing the number of alignment constraints), however it also drastically reduced the number of real phrases generated. Despite an improvement in the accuracy of phrase alignment, the overall quality of translation went down, so I did not use this method in my final results.

3 Parallel Decoding

Another problem I noticed with the baseline was very slow decoding. This was improved somewhat by reducing the size of the FST's generated, however I was able to achieve a much more drastic improvement by adding parallelization, allowing the decoding process to make full use of multiple processors.

4 Neural N-gram Model

I tried replacing the maximum likelihood n-gram model with an n-gram model generated from a feed-forward neural network. However, this did not improve performance. I suspect that this was because the neural network model used a relatively small vocabulary (limited to words that occurred at least 12 times due to computational constraints) whereas the maximum likelihood model was able to use the full vocabulary. With equivalent sized vocabulary, I think in theory the neural network model might be able to generalize better, but unfortunately that wasn't practical in this case.

5 Word-by-word Fallback

With the baseline model, I found that for some sentences no translation was produced. Often, this resulted from encountering a German word that was in the vocabulary but did not match any of the phrases generated. In order to prevent this issue, I added a word-by-word translation model using the translation probabilities generated by IBM Model 1. This word-by-word translation model was used as a fallback (with an appropriate penalty) to ensure that every word in the vocabulary had at least one valid translation.

6 Alignment Prior

The baseline IBM Model 1 encounters issues when the same word occurs multiple times in a sentence, since it has no way to decide which instance to use when generating alignments. I resolved this issue by adding a small penalty for alignments with large separations between aligned words. That is, I assumed that the i'th word in the source language sentence should align to approximately the i'th word in the target language sentence. Thus, when the best matched word occurs multiple times in a sentence, the model should select the one that is closest to the correct position. I initially applied this penalty both during training and when outputting the final alignments; however, I found that I got better results by applying the penalty only when outputting the final alignments.

7 Unknown Word Replacement

Of course, there will always be some unknown words. When producing a translation, I handled unknown words by inserting the original source language word untranslated. By assuming that the unknown words in the output would usually correspond to the unknown words in the input (in the same order), I was able to retrieve the corresponding source language word for each unknown word generated.

8 Phrase Count Plus 1 Smoothing

Finally, I found that many phrases only occurred once in the training data. The result was that these phrases had translation probability 1, which seemed to over-estimate the true confidence in these phrases. We would generally prefer to use a phrase that occurred many times rather than

one that occurred few times. To address this issue I added 1 to each denominator when calculating phrase probabilities, which seemed to improve performance.

9 Changes to Model Weights

I also adjusted the relative weights of the language model and the translation model. I found that decreasing the weight of the language model to 0.7 resulted in higher BLEU scores. I experimented with adjusting the interpolation constant between the unigram and bigram model, but I found that the original values ($\alpha_{unk} = 0.01$, $\alpha_1 = 0.1$, $\alpha_2 = 0.89$) performed better than the alternatives that were tested.

10 Result

Ultimately, I was able to achieve a BLEU score of 19.10 on the test set. This improves on the benchmark score of 17.98.

11 Conclusion

Various improvements were applied to the baseline phrase-based machine translation model including word-by-word fallback, improved alignment, unknown word replacement, and phrase count smoothing, resulting in a substantially improved BLEU score. The code can be executed using the script `./run-assignment.sh`.